

Skriptovanie v bashi

Bash (Bourne Again Shell) nie je len výborným príkazovým riadkom, ale aj plnohodnotným skriptovacím jazykom. Skriptovanie v shelly umožňuje zautomatizovať veľa úloh, ktoré by vyžadovali písanie veľkého množstva príkazov do príkazového riadku. Veľa programov v jednotlivých distribúciách Linuxu či Unixu sú len skripty príkazového riadku.

Bash je ako som už povedal skriptovacím jazykom, hlavný rozdiel medzi kompilovaným a skriptovacím jazykom je, že kompilované jazyky sú oveľa rýchlejšie a silnejšie ako skriptovacie jazyky. Tieto jazyky sa skladajú zo zdrojového kódu. Po napísaní tohto kódu sa program skompiluje a následne môže byť program spustený. Ale tento program je funkčný len na platforme na ktorej bol skompilovaný. Pre prechod na inú platformu je zdrojový kód nutné znova preložiť na prekladači inej platformy. (Např. Program v jazyku C pre Linux nebude fungovať vo Windows Vista).

Skriptovacie jazyky tiež začínajú písaním zdrojového kódu, ale už nie sú kompilované a spustené. Miesto toho interpreter jazyka číta inštrukcie zo zdrojového kódu a vykonáva ich, po prípade spúšťa programy, ktoré boli napísane do zdrojového kódu. Ale tým že interpreter číta inštrukcie po jednom je ich spracovanie výrazne pomalšie. Hlavná výhoda skriptovacích jazykov však spočíva v ich prenositeľnosti na iné platformy či operačné systémy.

Prvý program

Prvý program, ktorý si napíšeme bude už klasicky „Hello world“. Vytváranie zdrojového kódu začneme naštartovaním textového editoru a napíšeme doň tieto riadky:

```
#!/bin/bash  
echo "Hello, world"
```

Prvý riadok programu hovorí Linuxu aký sa má použiť interpreter. V našom prípade bash, ktorý sa nachádza v /bin/bash. Príkaz echo klasicky vypíše reťazec na obrazovku. Implicitne však textový editor nemá právo na spustenie, musíme ho teda pridať príkazom **chmod**. Predpokladajme že náš skript sa volá hello.sh (pripona sh nie je nutná). Príkaz vyzerá nasledovne:

```
chmod 700 ./hello.sh  
chmod +x ./hello.sh
```

Skript sa spúšťa nasledovne:

```
./hello.sh
```

Premenné

Ako v iných programovacích jazykoch sú premenné miesta v pamäti označené identifikátorom (názvom premennej). Tento identifikátor predstavuje pre interpreter adresu v pamäti, kde je uložená hodnota premennej.

V bash-i sú premenné deklarované nasledovne:

```
#!/bin/bash
x=5
echo "Hodnota premennej x je $x"
```

Vystup: *Hodnota premennej x je 5*

Z príkladu je vidieť, že k premennej sa pristupuje pomocou znaku "\$" uvedeného pred názvom premennej. Ešte poznámka pre tých ktorý sú zvyknutý písať medzery medzi = . V bashi nesmie byť pri = medzera.

Bash rozoznáva dva typy premenných:

- 1.) **Lokálne** - platné len pre konkrétny spustený skript
- 2.) **Systémové** – platné pre celý operačný systém, ktoré môžu byť využité všetkými programami systému

Premenné sú explicitne nastavované ako lokálne ak by ste chceli vytvoriť systémovú premennú môžete to urobiť nasledovne:

```
#!/bin/bash
export x=5
```

Alebo

```
#!/bin/bash
x=5
export $x
```

Systémové premenné sa dajú napríklad využiť na zistenie názvu interpreteru , ktorý je práve spustený:

```
echo $SHELL
```

Riadiace štruktúry

Bash obsahuje tieto riadiace štruktúry: *if, while, until, for, case*. Každá táto štruktúra je párová tzn. , že napríklad štruktúra podmienky začína *if* a končí *fi*.

If ... else ... elif ...fi

If je najbežnejšia riadiaca štruktúra, ako aj v iných programovacích jazykoch aj tu sa používa na vetvenie programu.

Príklad, ktorý nasleduje využíva vyhodnocovanie príkazom `test`, súbor `/etc/foo` skopíruje alebo ak súbor neexistuje vypíše chybové hlásenie. Príkaz `test` sa používa na otestovanie existencie súboru, príkaz na základe prepínača `-f` určí či `/etc/foo` je súbor. (Viac info *man test*)

Príklad:

```
if test -f /etc/foo
then #Soubor existuje. Zkopíruj tedy soubor
    cp /etc/foo .
    echo "Hotovo!"
else
    # Soubor neexistuje. Vypiš tedy chybu
    echo "Soubor neexistuje!"
    exit
fi
```

Poznámka:

```
if test -f /etc/foo
then
```

sa dá tiež zapísať takto:

```
if [ -f /etc/foo ]; then
```

While ... do ... done

Štruktúra *while* je štruktúra cyklu. V podstate vykonávanie určitého bloku príkazov dovtedy, kým je splnená podmienka za slovom *while*.

Príklad 1:

```
#!/bin/bash
while true; do
    echo "Press Ctrl-C to quit."
done
```

Príkaz *true* je program, ktorý vždy vracia hodnotu *true*. Nás skript bude teda neustále opakovať vypisovanie hlášky ako ukončiť program a to až do doby kým ho umelo neprerušíme (ctrl +c).

Príklad 2:

```
#!/bin/bash
x=0; # inicializuje hodnotu x na 0
while [ "$x" -le 10 ]; do
    echo "Aktuální hodnota x: $x"
    x=$((expr $x + 1)) # zvýšení hodnoty x o 1
    sleep 1 # čaká zadaný časový interval v tomto prípade 1 sekundu
done
```

V tomto príklade sme použili premennú *x*, ktorej hodnotu sme pri každom prechode cyklom zvýšili o 1 až kým nedosiahla hodnoty 10. Ako vidíte znova sme použili príkaz *test* (hranaté zátvorky), aby sme zistili či hodnota *x* je menšia alebo rovná 10. Ak je podmienka vyhodnotená ako *false*, program pokračuje za slovom *done*.

Príkaz *test* je veľmi užitočný preto uvediem niektoré možnosti jeho využitia:

Pre čísla:	Testovanie:
<i>x -eq y</i>	<i>x = y</i>
<i>x -ne y</i>	<i>x != y</i>
<i>x -gt y</i>	<i>x > y</i>
<i>x -lt y</i>	<i>x < y</i>

Pre reťazce:	Testovanie:
<i>x = y</i>	<i>x = y</i>
<i>x != y</i>	<i>x != y</i>
<i>-n x</i>	Vráti true, ak nie je reťazec prázdny
<i>-z x</i>	Vráti true, ak je reťazec prázdny

Until ... do ... done

Táto štruktúra je podobná štruktúre *while*, jediným rozdielom je, že cyklus beží dovtedy kým je podmienka vyhodnotená ako *false*.

For ... in ... do ... done

Štruktúra *for* sa používa, ak chcete behom cyklu použiť rôzne hodnoty pre jednu premennú a to tak, že pri každom priebehu cyklom bude mať premenná inú hodnotu.

Syntax: za príkazom *for* sa uvedie názov premennej, ktorá bude meniť svoju hodnotu a za slovom *in* jednotlivé hodnoty.

Príklad:

```
#!/bin/bash
echo "Zjišťuje se existence adresářů v systému"
for x in /dev /etc /bin /root /var /usr /home /mnt; do
  if [ -d "$x" ]; then
    echo "Adresář $x existuje"
  else
    echo "Adresář $x neexistuje"
  fi
done
```

Tento príklad testuje postupne, či v danom systéme existujú vybrané adresáre.

Case ... in ... esac

Štruktúra *case* je veľmi podobná štruktúre *if*. Štruktúra *case* je určená pre vykonanie nejakého kusu kódu podľa toho, akú hodnotu má premenná uvedená za príkazom.

Príklad:

```
#!/bin/bash
x=5;  # inicializuje x na hodnotu 5
# nyní se bude testovat hodnota x:
case $x in
  0) echo "Hodnota x je 0."
    ;;
  5) echo "Hodnota x je 5."
    ;;
  9) echo "Hodnota x je 9."
    ;;
  *) echo "Hodnota x není ani 0, ani 5, ani 9."
esac
```

Je jasné, že sa vykoná príkaz, ktorý je zapísaný za hodnotou 5. Ak nastane situácia, že nenastane ani stav 0,5,9 vykoná sa príkaz za znakom `*`.

Aritmetika v BASH-i

BASH ako aj väčšina programovacích jazykoch dokáže vyhodnocovať matematické výrazy. Ako ste už mohli vidieť aritmetika sa uskutočňuje pomocou príkazu *expr*. Ale príkaz *expr* je pomalý a preto sa aritmetika dá uskutočňovať pomocou interných príkazov *bash*-u. Alternatíva k výrazu *expr* je `$((...))`

Príklad:

```
#!/bin/bash
x=8
y=4
z=$(( $x + $y ))
echo "Součet $x a $y je $z"
```

Bash dokáže vykonávať nasledujúce matematické operácie:

Operácia	Znak
Sčítanie	+
Odčítanie	-
Násobenie	*
Delenie	/
Modulo	%

Poznámka: Bash dokáže pracovať len z celými číslami.

Čítanie užívateľského vstupu

Príkaz užívateľského vstupu je *read* . Jedná sa o vnútorný príkaz shell-u. Príkaz si vyžaduje ako parameter premennú.

Príklad:

```
#!/bin/bash
echo -n "Zadej své jméno: "
read jmeno
echo "Zdravím tě $jmeno!"
```

Funkcie

Ako aj v iných programovacích jazykoch aj v bash-i sa funkcie používajú na zjednodušenie a sprehľadnenie kódu. Príklad zápisu funkcie môžete vidieť nižšie

```
#!/bin/bash

hello()
{
    echo "Teraz sme zavolali funkciu hello()"
}
echo "Teraz sa zavolá funkcia hello()..."
hello
echo " Teraz sme zavolali funkciu hello()"
```

Samotná funkcia je určená svojím názvom po ktorom nasleduje dvojica zátvoriek (). Telo funkcie je uzavreté do zátvoriek {}. Funkcia musí byť definovaná ešte pred tým, než bude volaná. Pri deklarácii je tiež prípustné pred názov funkcie napísať *function* funkcia potom vyzerá nasledovne:

```
function hello()
{
    echo "Nyní jsme ve funkci hello()"
}
```

Funkcie môžu taktiež pomocou príkazu *return* vracať číselnú hodnotu. Ak je treba vrátiť reťazec, ten sa musí uložiť do premennej, ktorú po skončení funkcie môžeme použiť. Prípadne môžeme reťazec vypísať príkazom *echo* a výsledok odchytiť nasledovne:

```
#!/bin/bash
foo()
{
    echo Ahoj
}
...
result="$foo"
```

Vo vnútri funkcie môžete pomocou kľúčového slova *local* deklarovať lokálne premenné, ktoré budú existovať len v rámci zvolenej funkcie.

AND a OR

Zatiaľ sme používali len najjednoduchšiu formu podmienok, ak bude napríklad nejaký príkaz závisieť na viacerých podmienkach môžeme použiť znaky `&&`(and) alebo `||`(or).

Príklad:

```
#!/bin/bash
x=5
y=10
if [ "$x" -eq 5 ] && [ "$y" -eq 10 ]; then
    echo "Obe podmienky sú splnené"
else
    echo "Nie sú splnené obe podmienky"
fi
```

Použitie argumentov

Pri písaní programov sa často hodia určité parametre, podľa ktorých sa upravuje jeho beh. Tieto parametre samozrejme môžeme pridávať aj v bash-i. Tieto argumenty sú uložené v premenných. Premenná `$#` uchováva počet argumentov predaných programu. Jednotlivé argumenty sú potom uložené do premenných `$0,$1,$2,...`. Je teda vidieť, sú uložené v premenných číslovaných od 0 až po počet parametrov. V premennej `$0` je uložený názov programu. Parametrov môže byť spolu 9.